Regular Paper

Systems Integration: Key Perspectives, Experiences, and Challenges

Azad M. Madni[1, *] **and Michael Sievers**[2, †]

[1]*Daniel J. Epstein Department of Industrial and Systems Engineering, Viterbi School of Engineering, University of Southern California, Los Angeles*, CA 90089
[2]*Jet Propulsion Laboratory, California Institute of Technology, Pasadena*, CA 91109

Received 8 July 2012; Revised 29 September 2012; Accepted 29 September 2012, after one or more revisions
Published online in Wiley Online Library (wileyonlinelibrary.com).
DOI 10.1002/sys.21249

ABSTRACT

As systems continue to grow in scale and complexity, systems integration (SI) has become a key concern. This is especially the case in defense and aerospace. SI involves interfacing and enabling the interactions of component elements to collectively provide the functionality needed by the system to accomplish its goals. SI is part of the overall system development life cycle. SI increases in complexity when there are legacy systems that need to be integrated, and when humans are an integral part of the system. An added layer of complexity is introduced when the system has to exhibit resilience and adaptability in the face of contingencies in the operational environment. This paper addresses key perspectives and challenges in SI. Specifically, it presents the integration continuum, ranging from loose to tight integration. It presents a SI ontology that captures the key issues and concerns in a standard language. It also presents various categories of integration and their unique challenges. This paper is intended to clarify various types of integration and to stimulate new ways of thinking about SI. © 2013 Wiley Periodicals, Inc. Syst Eng 16: 000–000, 2013

Key words: systems integration; legacy integration; human-systems integration; semantic integration

1. INTRODUCTION

SI involves the efficient composition of components/subsystems into a whole that offers the required functionality and achieves specific goals [NASA, 2010; Madni, 2012b]. As systems continue to grow in scale and complexity, and the need for system adaptability increases [Madni, 2012a; Vuletid et al., 2005], *systems integration (SI)* becomes an increasingly more challenging proposition [Neches and Madni, 2012]. Specifically, defense and aerospace systems today have to satisfy a number of requirements such as affordability, reliability, adaptability, security, and resilience [Madni and Jackson, 2008; Neches and Madni, 2012; Neches, 2012; Rehage et al., 2004; Mosterman et al., 2005; Sage and Lynch, 1998]. These requirements further complicate SI.

SI is integral to the overall system development life cycle. A simplified SI process, based on the traditional waterfall model that has been used for decades, includes: *validation testing*, which focuses on whether or not the system performs the functions that are needed, and *verification testing*, which assures compliance with formally defined requirements. To the classical waterfall model flow, we have explicitly added the concept of "stakeholder," which goes beyond the tradi-

*Author to whom all correspondence should be addressed (e-mail: azad.madni@usc.edu.

† This work was done as a private venture and not in the author's capacity as an employee of the Jet Propulsion Laboratory, California Institute of Technology.

tional customer (Fig. 1). Stakeholders include the program/project manager, system project engineer, mission assurance engineer, verification and validation specialist, subsystem engineer, software engineer, electronics engineer, and human factors engineer.

A level in Figure 1 represents one of the traditional system elements such as "component," "assembly," and "subsystem." Each level potentially involves multiple, independent integrations. For example, a subsystem comprising a radio and a computer will be typically integrated and tested before being considered for integration at the next level.

Today, larger systems are frequently composed from a collection of unique solutions that typically results in highly customized, inflexible structures. Furthermore, the integration process tends to be inefficient and inflexible, especially when custom software and hardware are needed at each interface. Some systems employ *integration layers* to generalize the integration process, and assure flexibility for future extensions. An integration layer is an endpoint or interface between system components. For example, integration layers can be implemented as Web services, J2EE, Internet Inter-ORB Protocol, or by any similar technique that supports multiple protocols.

The integration flow for a *layered architecture* resembles the flow in Figure 1 in that, in a layered architecture, one or more applications are built up from lower-level components. However, there is a key distinction. Specifically, it is the integration layer that forms the basis of communication between applications, rather than applications communicating directly through custom interfaces. In this regard, designers need to evaluate factors such as how much data are expected to be shared and how fast the data need to be moved, and then decide which integration layer best addresses the needs and which protocols are best suited for each communications path.

Although integration layers provide adequate generality to standardize communications, the functionality is ultimately mapped to a physical or logical element of a hierarchical architecture. The service providing integration as well as the elements that communicate through it are assembled from subelements and tested to ensure that the expected functionality exists and the interfaces work as required. Since, layered integration makes use of standard, well-tested, and validated services and protocols, the quantity and type of testing that is performed is significantly reduced.

Yet another form of SI is *plug-and-play integration (PnP)*. In this form of integration, elements are added as and when they become available, and then are integrated into a system. PnP requires that the system and the elements that plug into it have an agreed to policy that allows the system to recognize a new element when it arrives and to have the needed software fabric for incorporating that element. While the manner in which integration occurs differs somewhat from the classical flowdown, it is once again necessary to build up and test PnP capability in terms of satisfying required subsystem functionality.

There are a few other integration methods that have been in use for some time. These include: star integration (all subsystems are connected to each other); horizontal integration (dedicated subsystems implement a bridge between other subsystems); and build integration (frequent, usually automated software build and test). While these methods are rooted in the classical hierarchical integration approach, they differ in the testing mechanisms employed.

Over the past decade, SI is being increasingly viewed as a source of competitive advantage [Prencipe et al., 2011; Best, 2003] in a wide variety of sectors such as computing, automotive, telecommunications, military C4ISR, and aerospace. At its core, SI is a strategic activity that is integral to business management and that cuts across technical, management, and strategic levels. Key to smooth operation and dynamic adaptation of systems, SI can be viewed at different levels and from a variety of perspectives including lifecycle, architecture, process, interface, enterprise, product, and data [Abel et al., 1994; Biffl et al., 2009; Hobday et al., 2005]. Communication

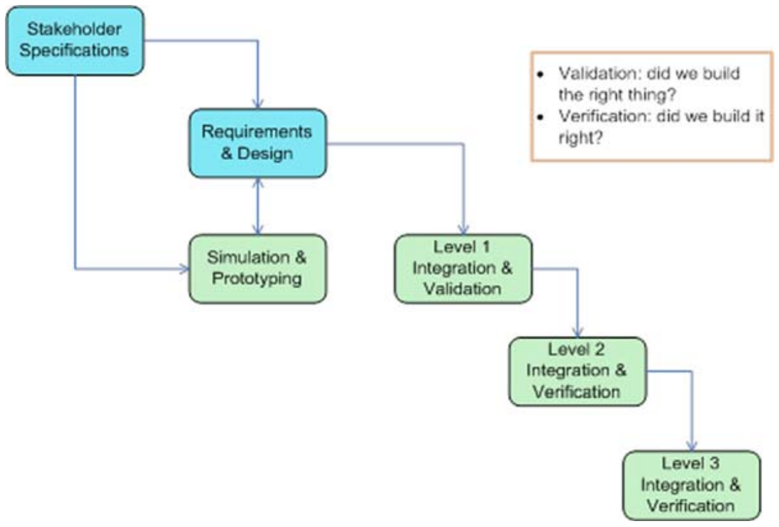

Stakeholder Specifications
Requirements & Design
Simulation & Prototyping
Level 1 Integration & Validation
Level 2 Integration & Verification
Level 3 Integration & Verification
• Validation: did we build the right thing?
• Verification: did we build it right?

Figure 1. Simplified system integration (SI) process. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

is central to successful SI and at the root of SI failure. We use the term “communication” broadly to mean information transport within a system as well as the interactions among system stakeholders. To enable effective communication, we offer an SI ontology that addresses both forms of communication with the goal of minimizing errors that cause SI failures.

The remainder of the paper is organized as follows. Section 2 defines SI and discusses SI challenges. Section 3 presents examples of SI failures and their possible causes. Section 4 discusses integration from a variety of perspectives and presents an SI ontology to inform and guide SI. Section 5 discusses the cross-cutting nature and the pros and cons of interoperability. Section 6 discusses key issues in legacy systems integration and discusses various approaches for integrating legacy systems. Section 7 discusses existing approaches to human-systems integration and emphasizes the importance of closed loop concept engineering. Section 8 summarizes the key contributions of this paper.

2. SI CHALLENGES

System Integration (SI) is concerned with forming a coherent whole from component subsystems (including humans) to create a mission capability that satisfies the needs of various stakeholders. The components, once developed, are integrated and ultimately tested in the deployment environment. There are different forms of SI. Historically, vertical integration occurs when components of a system developed by a single acquisition program are integrated to produce the desired capability. Today, vertical integration has a broader meaning which covers both a single organization (e.g., US Navy) and joint organizations which have multiple acquisition programs over time that contribute to the creation and enhancement of a mission capability (e.g., net-centric surface-to-surface combat). Horizontal integration occurs when systems developed by different acquisition programs and often for different customers and purposes, are brought together to create a new capability.

Figure 1 depicts an ideal, well-controlled, rigorous process. However, as currently practiced, SI remains largely an ad hoc process that primarily revolves around the needs of the stakeholders, technologies involved, and legacy constraints [Madni, 2012a]. For example, cost and schedule constraints often conflict with the need to conduct thorough testing at a given integration level before moving up to the next level. Consequently, lower-level problems propagate up the integration hierarchy where they become more difficult and expensive to isolate and correct. Occasionally, a subset of resources for a given integration may be unavailable. Rather than wait for the resources, management may decide to begin partial integration with the available resources and accept the risk that when the missing elements arrive, their incorporation will not invalidate the partial integration. Several real issues arise when integrating with legacy systems. Consider integrating with a system that is relatively old and has little documentation that details its limitations and capabilities. For example, a legacy system may have been designed to process up to some fixed number of concurrent transactions. However, the new system might be required to the handle a greater number of transactions. If that limit is not known to system designers and integrators, then unexpected behaviors are likely to show up that might require substantial debugging and corrective effort.

While SI has been narrowly defined in some industries (e.g., medical devices) by rigorous methodologies that achieve specific goals, the suitability of these methods to other industries has not been found to be practical. NASA, for example, requires very detailed procedures, testing, failure reporting, review boards, and documentation throughout SI. This rigor is justified by system cost, mission complexity, and public perception of failure. In other industries, the impact of delaying the release of a system is considered more important than assuring that integration has been thoroughly tested. In particular, in some of the so-called “agile” design methodologies, delivery speed trumps stability, functionality, and robustness. In other words, the stakeholders in the different industries have differing goals that drive both the SI approach and the level of rigor. A further complication is the lack of a common language for communicating among disciplines. The situation is much the same when it comes to system descriptions motivated by architectural frameworks such as DoDAF, modeling languages such as SysML and UML, and domain-specific languages and ontologies.

SI is intimately linked to interface testing, validation, and verification [Stavridou, 1999; Siemieniuch and Sinclair, 2006; Rossak and Ng, 1991; Nilsson et al., 1990]. The history of SI reflects the growth in system complexity and performance requirements over time. The definition of the term “system” itself has evolved over the years. In the past, systems were simple, closed, and well-defined (e.g., aircraft, automobile). Today they are also complex, open, ill-structured and dynamically-changing (e.g., the Internet, healthcare, power/energy grids). Existing systems integration tools no longer suffice for such systems. For example, in a closed system a network monitor can develop a good statistical model for normal traffic and signal when the observed traffic falls outside the normal range. In an open, complex system, it may not be possible to distinguish normal from unusual traffic because network usage varies wildly, depending on the functions being implemented and the connectivity that is required.

In light of the foregoing discussion, there is a pressing need for new ways of thinking about SI. The next two sections examine integration failures and discuss SI from several key perspectives to illuminate the key challenges that system integrators face today [Bonjour and Falquet, 1994; Luo et al., 2006; Horowitz et al., 2003; Lee et al., 2003; Halevy et al., 2005].

3. INTEGRATION FAILURES

A key goal of SI is to assure that semantic and syntactic interfaces between component elements of the system perform as specified by “contracts” between the elements. A companion goal is to assure that the interfaces can be adapted in well-understood ways with relatively modest effort. A semantic interface addresses the meaning and use of information while a syntactic interface provides the means by which semantic content is delivered. By "contract" we mean that customers write what they want built (in the form of a specification, work order, requirements document, or interface control document) and developers agree to build it. While it is possible to develop, integrate, and test trivial systems without written documentation, SI for nontrivial systems can be expected to fail without written documentation because customers and developers cannot identify and exchange pertinent information often enough to avoid mistakes.

Almost all SI failures occur at interfaces primarily due to incomplete, inconsistent, or misunderstood specifications. Invariably, the root causes of failure tend to be ad hoc integration and failure to develop and adhere to formally defined semantic concepts and relationships. Examples of well-known failures resulting from lack of careful SI planning and execution are:

a. *Ariane 5 Launch Vehicle Failure*. In 1996, the first Ariane 5 launch vehicle was destroyed when the backup inertial reference system (IRS) failed immediately before the failure of the primary IRS. The trajectory profile for the Ariane 4 was loaded into the IRS software, which resulted in a software exception in the conversion of floating point values to integers. The IRS software was analyzed and operations involving seven variables were at risk of leading to an operand error. The analysis led to protection of four of the variables but three, including the Horizontal Bias (BH), were left unprotected because they were physically limited or assumed to have a large safety margin. No testing was done to verify these assumptions.

   The launch vehicle began disintegrating about 39 seconds after launch due to high aerodynamic loads caused by an angle of attack of more than 20°. The loads caused the boosters to separate from the main stage, which triggered the self-destruct mechanism. The improper angle of attack was caused by full nozzle deflections of the solid rocket boosters and the main engine. The nozzle deflections were commanded by the Onboard Computer software on the basis of data transmitted by the active IRS. Part of these data contained a diagnostic bit pattern which was interpreted as flight data. The active IRS did not send correct attitude data because it had declared a failure due to a software exception in the BH variable. Because the backup had already failed, the Onboard Computer could not switch over to the backup. The Ariane 5 failure was the result of a semantic interface error that propagated into a critical variable that was not protected against unexpected values. A complete failure review is available at: http://www.di.unito.it/ ~damiani/ariane5rep.html.

b. *Airbus Production Delay*. An integration problem delayed the Airbus A320 delivery by 2 years and cost Airbus more than $6 billion. The cabin wiring harnesses delivered by a German factory to the French assembly line didn't fit properly into the plane. The problem resulted from a mechanical description used by the Germans that was incompatible with the mechanical specification of the actual aircraft. This failure had a semantic interface error as its root cause that resulted in a syntactical interface error in the factory—i.e., an error in physical interfaces [Kilroy, 2006].

c. *USS Yorktown Computational Failure*. In 1996, the USS Yorktown was outfitted as a testbed for "smart ships." An integrated control center was implemented using a 27-processor network communicating over a fiber-optic cable. In 1997, a zero inadvertently entered into a database field resulted in a divide-by-zero error which brought down all of the machines on the network. This failure caused a failure of the propulsion system which required that the ship be towed back into port. In this situation, interfaces were not fully characterized and requirements were not properly imposed on failure propagation. This error is an example of an ad hoc design that was not thoroughly vetted in its expected operational environment (http://gcn.com/Articles/1998/07/13/Software-glitches-leave-Navy-Smart-Ship-dead-in-the-water.aspx).

The foregoing examples lead to an important question: Could a systematic integration methodology have prevented such failures? Given the growing complexity of today's systems and the impracticality of exhaustive testing (i.e., testing under all possible operating conditions), no methodology can guarantee totally error-free integration and operation. However, given that integration is primarily a communications problem, the development and adherence to a systems integration (SI) ontology can inform and guide system design and integration, ensure consistent communication among stakeholders, and enforce consistent physical and logical interfaces within the system.

4. KEY SI CONSIDERATIONS

SI cuts across a number of factors that collectively contribute to the complex underlying issues [Krygiel, 1999; Browning, 2001; Cummins, 2002; Maier and Rechtin, 2002; Halevy et al., 2005; Madni, 2010, 2011). Each factor influences and, in some cases, controls how SI is accomplished and when it is considered successful. Each key factor is discussed next.

4.1. Integration Semantics

There are several factors that affect the different levels of SI [Mayk and Madni, 2006]. Collectively, these factors provide the semantic underpinnings for defining and managing SI. A unified view of these factors is presented in the SI ontology (Fig. 2). The SI ontology offers a means for representing the concerns and expectations of SI in a unified manner. In particular, by considering each element of this model, integration can proceed along a path that minimizes miscommunications. More importantly, it fills a void in the ability to bridge system complexity, interface problems, and system representation. Specifically, the SI ontology provides the basis for building a checklist that can be reasoned with in ways that allow many integration problems to be avoided or detected and circumvented. At all levels, SI includes artifacts (documentation) and metrics (measurements used to assess integration success). For example, SI concepts in the SI ontology (Fig. 2) inform us that SI is composed of Certifica-

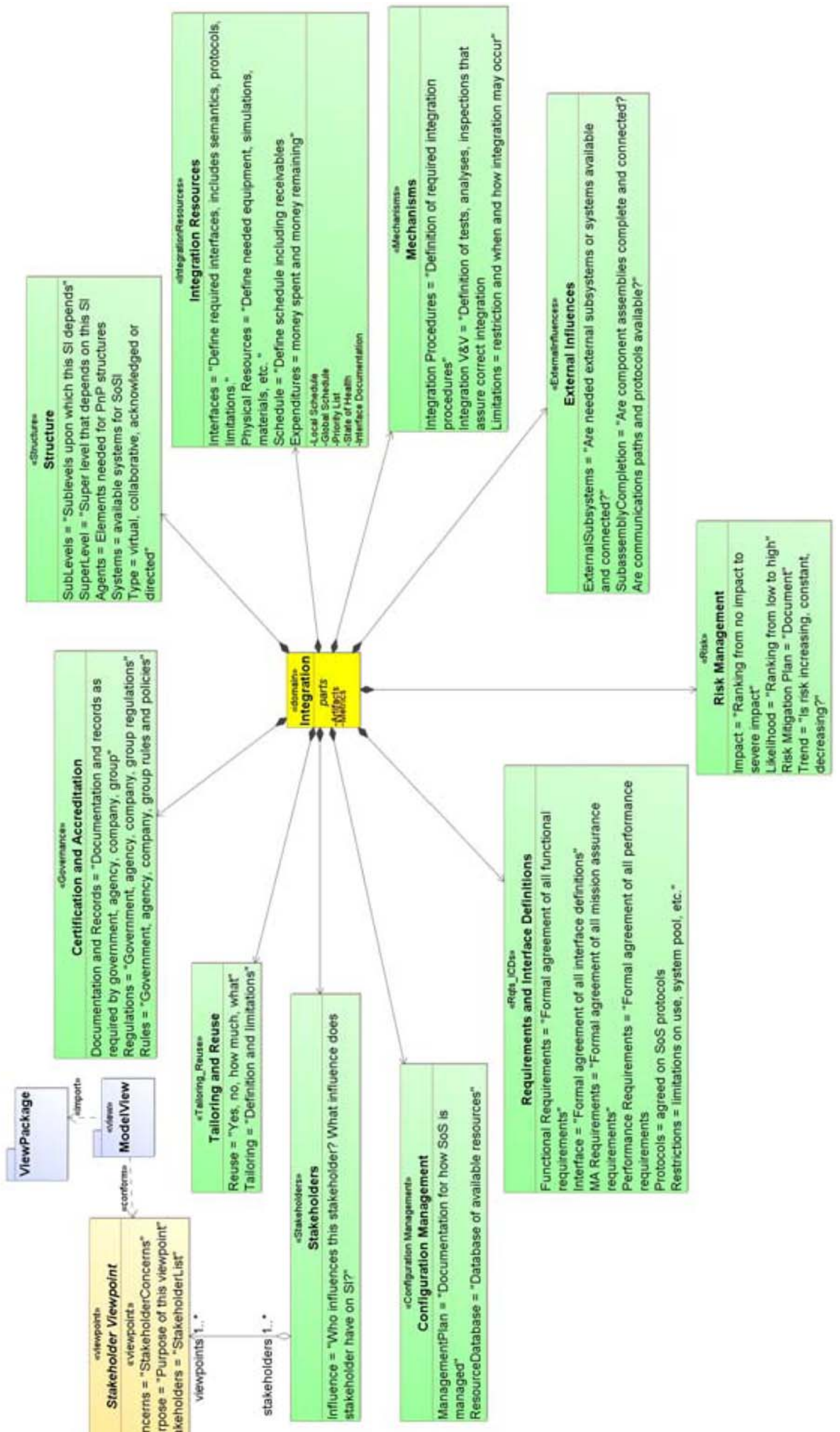

«domain» Integration
parts
«artifact» Items
«Structure» Structure
SubLevels = "Sublevels upon which this SI depends"
SuperLevel = "Super level that depends on this SI
Agents = Elements needed for PnP structures
Systems = available systems for SoSI
Type = virtual, collaborative, acknowledged or directed"
«IntegrationResources» Integration Resources
Interfaces = "Define required interfaces, includes semantics, protocols, limitations."
Physical Resources = "Define needed equipment, simulations, materials, etc."
Schedule = "Define schedule including receivables
Expenditures = money spent and money remaining"
-Local Schedule
-Global Schedule
-Priority List
-State of Health
-Interface Documentation
«Mechanisms» Mechanisms
Integration Procedures = "Definition of required integration procedures"
Integration V&V = "Definition of tests, analyses, inspections that assure correct integration
Limitations = restriction and when and how integration may occur"
«ExternalInfluences» External Influences
ExternalSubsystems = "Are needed external subsystems or systems available and connected?"
SubassemblyCompletion = "Are component assemblies complete and connected?
Are communications paths and protocols available?"
«Risk» Risk Management
Impact = "Ranking from no impact to severe impact"
Likelihood = "Ranking from low to high"
Risk Mitigation Plan = "Document"
Trend = "Is risk increasing, constant, decreasing?"
«Governance» Certification and Accreditation
Documentation and Records = "Documentation and records as required by government, agency, company, group"
Regulations = "Government, agency, company, group regulations"
Rules = "Government, agency, company, group rules and policies"
«Rqts_ICDs» Requirements and Interface Definitions
Functional Requirements = "Formal agreement of all functional requirements"
Interface = "Formal agreement of all interface definitions"
MA Requirements = "Formal agreement of all mission assurance requirements"
Performance Requirements = "Formal agreement of all performance requirements
Protocols = agreed on SoS protocols
Restrictions = limitations on use, system pool, etc."
«Tailoring_Reuse» Tailoring and Reuse
Reuse = "Yes, no, how much, what"
Tailoring = "Definition and limitations"
«Stakeholders» Stakeholders
Influence = "Who influences this stakeholder? What influence does stakeholder have on SI?"
«Configuration Management» Configuration Management
ManagementPlan = "Documentation for how SoS is managed"
ResourceDatabase = "Database of available resources"
ViewPackage
«import»
«view» ModelView
«conform»
«viewpoint» Stakeholder Viewpoint
«viewpoint»
concerns = "StakeholderConcerns"
purpose = "Purpose of this viewpoint"
stakeholders = "StakeholderList"
viewpoints 1..*
stakeholders 1..*

Figure 2. SI ontology (a semantic model). [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

tion and Accreditation, Structure, Integration Resources, Mechanisms, External Influences, Requirements and Interface Definitions, Configuration Management, Stakeholders, and Tailoring and Reuse. It is important to note that the Stakeholders share a number of Viewpoints (or perspectives). Using SysML constructs, each Viewpoint can be associated with a View, which is a specific window into the system. A View may import any number of aspects of the system description as needed to support the View.

SI activities can begin by detailing the specifics for each of the elements defined in Figure 2. That is, system stakeholders need to establish the boundaries, expectations, limitations, resources, requirements, responsibilities, test conditions, milestones, deliverables, receivables, and approaches by fleshing out each element relevant to SI. Along with schedules and budgets, the SI ontology can guide the SI process and facilitate communication among the stakeholders. The key concepts represented in Figure 2 are discussed next.

4.2. Influence of External Factors

Determining when SI activities can actually begin depends on external factors that are closely related to the required integration resources. Integration Resources define the "what" of SI, while "external factors" define influences that affect the "what." Thus, Integration Resources are the technical, management, budgets, and agreements necessary for integration. External Influences potentially contribute to Integration Resources.

For example, a system implementation might depend on a particular protocol defined by an international standards organization. Although the system integrator may have some influence on the decisions of the international body, the integrator may not be able to change aspects of the protocol that are inconvenient or inefficient for the particular mechanisms planned for the system. In this case, the international body is an external influence on the design and integration of the system.

There can also be nontechnical external influences that affect SI. For example, suppose a system needs to be installed by a certain date. Suppose also that if that date is missed, the next available opportunity could be weeks or months later. In this situation, the system integrator needs to assess the risks of missing the installation deadline, and formulate risk mitigation plans accordingly.

4.3. Structure

Historically, integration is a hierarchical process in which structure defines the influences that apply at a specific integration level. That is, the components are structured at a particular level and then at the level above (i.e., the level at which the components are expected to be integrated).

In SI, subelements are typically designed to have compatible communications and semantics. The key is verifying that these subelements correctly implement their interface contracts. Other options for SI include continuous integration (build integration) and plug-and-play integration. In continuous integration (CI), members of a team submit their inputs to a build process that is often automated. CI is most often associated with software releases, but could just as well apply to reprogrammable hardware such as FPGAs. After each new release is created, it is subjected to tests that assure that all previous functions still work (regression testing) and all new features are correctly implemented. For CI to work effectively, the system is structured in place with tools that permit rapid builds and tests.

As noted earlier, SI can potentially follow a number of paths that suit the specifics of the system being integrated, the stakeholders involved, and conventional practices within an industry. For example, plug-and-play (PnP) integration requires that the system elements adhere to a set of rules that enables a system to identify the elements and the required software needed for communication. Software agents, for example, can enable integration of heterogeneous devices as described for a medical application in Cabri [2007]. PnP requires a structure in which agents or equivalent capability are deployed and managed as elements are added or removed. Similarly, integration layers imply yet another structure that has a defined set of services and protocols. Whether SI proceeds in the classical hierarchy or one of the many other options, structure is key to understanding the functional allocations and communications that need to exist and be tested.

4.4. Requirements and Interface Definitions

Requirements and interface definitions are formally defined as verifiable agreements that specify functionality and performance requirements within and between system components. The definitions of requirements and interfaces result from a flowdown of user and customer needs through a process of refinement and allocation. Each requirement and interface is verified through a process that typically involves test, analysis, inspection, and/or demonstration. Invariably, other system elements influence requirements and interfaces.

In our approach, "requirements" and "interface definitions" are treated in the same way using common semantics. This is because the two are intimately linked and quite often viewed as the same because they both represent binding contracts. Requirements are a contract between a developer and those stakeholders that interact with the developer. Interfaces enable communication and interaction between elements within a system as well as between the system and its environment. In the interest of simplification, we employ the term "requirements" from this point on as short for both *requirements* and *interface definitions.*

We explicitly include an "interface property" in our SI ontology. This property is a stereotype that can be applied to communication ports in a SysML representation of the system. The stereotype has four properties: semantics, syntax, units, and limits. This stereotype requires designers to explicitly identify key properties that most likely can cause integration problems if not consistent. To this end, reasoning tools can be used to verify interface consistencies. Since internal system interfaces are generally specified and designed to assure compatibility, the interface property is most useful in evaluating compatibility of external interfaces. It is important to note that the interface property applies equally to process flows. In the A320 production delay problem discussed above, the inclusion of this property would have discovered the discrepancy between the aircraft specification and the mechanical descriptions used in the factory.

Requirements are precise, verifiable statements of what a system needs to provide to achieve its goal. Requirements may be in the form of written text, use cases (behaviors), functional design descriptions, interface control documents, and other such artifacts that best suit the particular style of a given industry. Regardless of the form, requirements inform designers what they are expected to deliver and test. All too often, though, requirements are neither precise nor verifiable. Worse yet, they may not even be aligned with system goals, or they may be in conflict with other requirements. In these circumstances, integration inevitably fails adversely impacting development and test schedules and costs.

Ideally, all requirements should flow from definitions of system usage and constraints. In particular, higher-level requirements are decomposed into lower-level allocations with each design iteration. This process eventually leads to a level of specificity that designers can implement. The implemented components are tested against requirements and assembled into higher-level elements which are tested against their corresponding requirements. In the well-known "V" diagram, decomposition of requirements occurs on the left of the "V," while the buildup and test of components occurs on the right of the "V."

However, there are several real world considerations that conflict with the ideal flowdown pattern in the traditional "V" diagram. First and foremost, real systems are too complex and have too many fuzzy aspects to be amenable to pure top-down decomposition. Consider the situation where a user wants a "simple interface" but does not have the requisite knowledge to specify what "simple" means. In this situation, human factors engineers have to produce multiple interface options during system design for the user to evaluate and choose from. All too often, a stakeholder may request that the system provide a certain feature (e.g., a capability or a function). Systems engineers decompose that feature into lower levels. At the implementation level, it may turn out that the

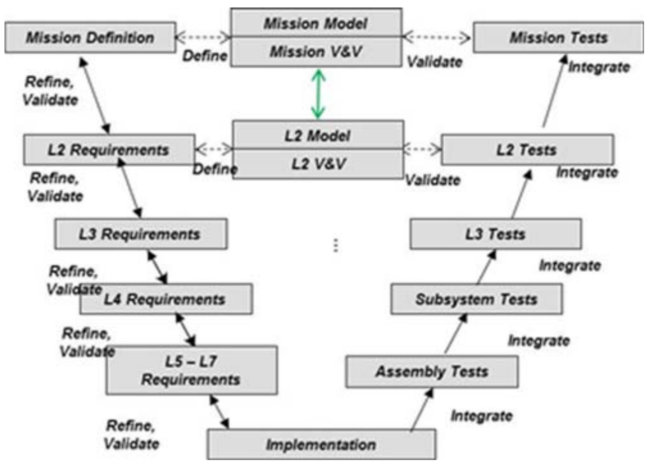


Figure 3. Modified "V" requirements flowdown with intermediate validation. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

requested feature is cost-prohibitive. If the stakeholder is unwilling to pay for the implementation, then an iterative upward process is undertaken in which a less costly alternative is found to provide the feature, the requirement for the feature is modified, or the feature is "dropped" from the design. Partly in recognition of this issue, the aerospace industry has instituted the concepts of *objective requirement* and *threshold requirement.* The objective requirement is akin to a stretch goal, while the threshold requirement is the actual goal. For example, the F-18EF had an objective requirement of Mach 2 and a threshold requirement of Mach 1.6. The engineering team satisfied the threshold requirement of Mach 1.6, and later on was able to achieve the objective requirement of Mach 2.

In the classical "V," testing is conducted on the right side as lower-level components are built up and integrated into higher-level components. In some cases, this means that the implications of testing a given requirement are not considered until quite late in the process. Just as a feature may prove to be too costly to implement, the test for a feature may also prove to be too costly to implement. If it turns out that the design has progressed to the point that the feature has already been built, then the stakeholders need to decide whether to pay for the test, pay for a less precise test, accept analyses results, accept the risk that the feature might not work, or disable the feature. If the cost of a test is known early on, then stakeholders can go through the same upward iterations to refine the feature, find a cost-effective test, accept a calculated risk, or "drop" the feature from the design.

Another drawback of the classical "V," is that errors in flowdown of requirements to lower levels may not be detected until higher levels of integration. The cost of finding and fixing design errors grows dramatically with increasing levels of integration. The key point here is that the process needs to become more agile. The agile process shown in Figure 3 is designed to circumvent this problem.

Quite often, peers on an engineering team associated with a particular level in the system hierarchy collaboratively create requirements for how best to allocate responsibilities related to a higher-level requirement. Generally, these requirements are approved by higher-level stakeholders. However, the process does not strictly adhere to the classical flowdown. For example, consider the case where a controls engineer is given a higher-level requirement to stabilize a specific mechanical process to a certain level. Accomplishing this stabilization could require reading multiple sensors, implementing a control algorithm, and adjusting actuators at a given rate. The controls engineer needs to work with electronics and software engineers to establish interface and performance requirements that are compatible with the controls bandwidth needed to satisfy the stability requirement. Finally, requirements may be self-derived at a given design level. Self-derived requirements are requirements without a parent. However, the designer at a given level is cognizant that they must be included as part of the implementation. Self-derived requirements today are treated as valid requirements that carry the same weight as requirements that have been flowed down, and have the same need for formal testing as flowed down requirements.

As noted in the Introduction and in Section 4.3, even though a system might follow a nonhierarchical integration implementation, there are still requirements that drive the subelements that need definition and evaluation. The form of those requirements and the resulting tests performed will vary for nonhierarchical integration; however, the basic structure shown in Figure 3 still applies. For example, at the top-level a stakeholder may specify that the system be integrated with a particular form of integration layer. At lower levels in the "V," requirements are allocated to subelements that come together to provide the integration layer or communicate through it.

4.5. Certification and Accreditation (C&A)

C&A defines the records, tests, regulations, rules, policies, and other such considerations that are binding on an integrated system. For example, a company may write a policy that defines how components of a system must be grounded. All products produced by that company need to adhere to that policy to ensure that integration does not fail due to grounding-related incompatibilities. The form that the C&A takes depends on the particular use of the system and the impact of failure. For example, in the aircraft industry, C&A takes the form of FAA safety regulations and test, and internal policies. In the medical industry, C&A takes the form of FDA safety regulations and test and internal policies. In the banking industry, C&A takes the form of internal bank policies, backup, and disaster recovery policies. In the space industry, they comprise NASA requirements or equivalent safety requirements and test and internal best practices. In the nuclear power plant industry, C&A takes the form of NRC safety requirements and tests, and internal policies.

4.6. Stakeholders

Stakeholders are individuals, organizations, or external systems with an interest in the system. Stakeholder "concerns" are those issues that are important to a particular stakeholder. Stakeholders can exert "influence" on some aspect of system integration, and can, in turn, be influenced by other aspects. The concerns and influences of a stakeholder define a "viewpoint" or perspective of the system important to that stakeholder, i.e., those aspects and relationships that the stakeholder needs to exert influence on and be influenced by. Table I presents typical stakeholders and their role in system integration.

In many ways, SI requires defining stakeholders, and then properly managing their needs and expectations. Significantly, the needs and expectations of some stakeholders are likely to be in conflict with those of other stakeholders. Consider, for example, a manager concerned with keeping a project on schedule and a customer who wants to add a capability that is out of scope. Managing stakeholders' expectations is as much a part of SI as the process of building up and testing the system.

Table I. SI Stakeholders, Concerns, Influences, and Needed Resources

Stakeholder	SI Concerns	Influences	Metrics	Needed Resources
Project Manager	Cost and schedule for performing SI	• Input: System customer • Output: Funding priorities	Earned value, schedule	Funding
System Project Engineer	Performance, behaviors, operability, physical, electrical restrictions, and functionality of integrated system	• Input: Project Manager • Output: Subsystem Engineering priorities and allocations	Requirement satisfaction, performance parameters	Customer use cases, functioning systems (subsystems), tested interfaces
Mission Assurance	Safety, reliability, contamination, physical stress tolerance of integrated system	• Input: Customer's environment • Output: Component selection, EMI/EMC, grounding, personnel and equipment safety, test requirements,	Temperature range, total dose, shock/vib, FMECAs & FTAs, EMI/EMC susceptibility.	Thermal analyses, shock and vib test results, reliability test results
Validation and Verification	Assurance that integrated system meets user needs (validation) and that the system is correctly designed and meets requirements (verification)	• Input: Mission goals and formal requirements • Output: Test definition, test execution and result certification	Performance characteristics, measure of test completion	Integrated system, test equipment, test software
Subsystem Engineer	Ensure subsystem interfaces works with other subsystems and human roles	• Input: System constraints and performance requirements • Output: Compliant design	Physical variables in range, satisfy performance requirements	Subsystem test workbench, test software, MCAD tools
Software Engineer	Ensure software interfaces with hardware and human roles	• Input: Hardware and human constraints • Output: Compliant design	Software behavior is predictable	Software development and test environment
Electronics Engineer	Ensure electronics are "correct by construction and compatible with subsystem requirements	• Input: Subsystem constraints • Output: Compliant design	Electronics satisfy interconnection and thermal requirements	EDA tools
Human Factors Engineer	Ensure that human interactions with subsystems are consistent, inspectable, with manageable cognitive load and acceptable error rates	• Input: interaction requirements, Human System Integration (HSI) constraints • Output: Manageable cognitive load, acceptable error rates	Human acceptance (usability, utility)	HSI tools, MMI prototyping tools

Table I implies a common theme (pattern) for all SI stakeholders. As shown in this table, each stakeholder has one or more concerns that define the scope of their interest. All stakeholders are influenced by and can influence one or more stakeholders. Some stakeholders produce artifacts for their use and for the use by other stakeholders. All stakeholders produce and monitor metrics that measure system parameters associated with their concerns. It should also be evident that the interactions, associations, and semantics that link stakeholders are themselves communications pathways, checks and balances, and (if properly constructed) testable assertions.

4.7. Mechanisms

Mechanisms are processes, procedures, tests, and inspections that are required for integration and that are motivated by safety and verification concerns. At the most basic level, mechanisms assure that appropriate safety precautions have been taken (e.g., before attaching a cable to a connector). At higher levels, procedures are put in place to ensure that enabling a given function for the first time will not adversely affect or cause the failure of other components, subsystems, or systems. It is important to note that mechanisms tend to be somewhat specific to given industries and products. Table II presents some common mechanisms used in spacecraft integration. These mechanisms are fundamentally applicable to any integration process. It is also worth noting that each mechanism has associated documentation that gets filed (i.e., persistently stored) in the development project database. These documents serve as evidence that certain required processes were in fact performed. As important, they become a source of information that can be useful for diagnosing anomalous behavior after a system has been deployed.

4.8. Integration Resources

Integration resources include the interfaces, test equipment, models, simulations, online documentation, and FAQs required for performing a given integration step. The specific integration resources employed are a function of integration content and system life cycle phase. Integration resources help with integration concerns such as hardware-software integration, human-systems integration, and legacy integration. As previously noted, Integration Resources are related to External Influences.

There are other circumstances in which an external system (or subsystem) cannot be made available during certain stages of integration because of safety or cost concerns. In such situations, appropriate fidelity simulations are the only viable alternatives that need to be exploited as stand-ins for the unavailable elements. In these cases, the simulation needs to exhibit the right behavioral fidelity from, for example, electrical, functional, and temporal perspectives. In some industries this can be a tall order often requiring development activities that can potentially turn out to be almost as expensive as the entity being simulated. Fortunately, this is not the case in most industries. On the other hand, if the simulated entity is not available, then either the integration needs to wait until such time that it is, or the integration is performed using models and simulations of the real world entity.

With respect to the latter, spacecraft systems are often integrated using precise, validated simulations of the unavailable subsystems. The key point here is that these simulations have gone through validation. In this context, validation is a formal process that compares the simulated behavior of a system to the behavior of the real system to assure that the simulation is an adequate substitute. The cost and complexity of validation can be reduced if both the simulation and the actual system are derived from the same set of requirements and models. A recent JPL project accomplished its validation goal through a detailed simulation based on SysML models of a spacecraft command and data handling system [Kahn et al., 2012].

4.9. Other Key Considerations

Tailoring and Reuse. The tailoring and reuse of system components and processes are an integral part of integration activities. The degree of tailoring and the amount of reuse possible varies from one system/problem to another. Program requirements drive the amount of reuse that is realistically achievable given legacy and technology constraints. Program requirements also determine how much tailoring is needed in the system life cycle processes and the system components.

Table II. Key Mechanisms and Their Purposes

Mechanism	Purpose
Integration Planning Review	Review of test sets, staffing, scheduling, and test flow definition
Testbed Testing	Includes simulation and simulation model validation as well as testing on specialized testbeds.
Mechanical Testing	Assembly and testing of structures and mechanisms.
Electrical Integration and Functional Testing	Interface, electrical, grounding, and functional testing.
Environmental and Reliability Testing	A collection of tests that assure the system operates properly under the environmental conditions it will be deployed into (temperature, radiation, shock, vibration, vacuum, electromagnetic compatibility, etc.)
System Integration Review	Evaluation of a project's plans for additional development and readiness to start assembly, perform testing, and commence operation once deployed. This review also evaluates project validation and verification requirements and plans.

Once this determination is made, it becomes possible to ascertain whether or not a planned implementation is feasible (i.e., can be completed). Clearly, the requirement to use legacy subsystems limits the degree to which tailoring and reuse are potentially possible. In DoD systems, it is not unusual to find up to 70% legacy subsystems [Neches and Madni, 2012].

Risk Management. Risk management (and configuration management) activities are an integral part of SI initiatives. A risk may be conceptualized in terms of an event and impact, for example: If A occurs, then it causes B. Some industries quantify risk with two parameters: The likelihood that a given event will occur and the impact of that event. The DoD and NASA, for example, create two-dimensional risk charts that use the likelihood of an event as one axis and the impact of the event as the other axis. The intersection of high likelihood and high impact denotes a serious problem, while low likelihood and low impacts are less important. Various combinations in-between indicate serious, moderate, and minor problems. Generally, risks are tracked and "statused" at reviews, and may require risk mitigation plans for retiring the risk or moving it to a lesser level of importance. Risk management during SI subsumes acceptance testing, deployment, transition from existing system to new system, operations, maintenance, logistics, training, and upgrades.

Integration Plans. SI needs integration plans which are derived from development plans, engineering plans, and test plans.

5. SYSTEMS INTEROPERABILITY IS A CROSS-CUTTING ISSUE

A system is seldom totally stand-alone entity. Typically, systems provide services to users who might have direct connections to or connect indirectly through other systems. The SI process needs to accommodate these external influences and resources through system design and the selected integration mechanisms. In this regard, interoperability drives integration resources, external influences, structure, and mechanisms identified in the SI ontology. The SI ontology (Fig. 2) informs and guides the design of interoperability among subsystems.

Interoperability is the ability of distinct systems to share semantically compatible information and then process and manage that information in semantically compatible ways, enabling users to perform specific activities [Ziegler and Mittal, 2008]. Interoperability applies to disparate systems that were not originally designed with interoperation in mind. *Interoperability* is a cross-cutting concern that relies on agreements on concept labels and whether or not a particular "label" means the same thing or different things. Therefore, interoperability is beyond the scope of a single system development effort, or organization [Naudet et al., 2010]. The purpose of interoperability is the creation of a capability to satisfy a mission requirement, or an important human purpose. Thus, interoperability is more than getting systems to communicate with each other. It requires that the organizations involved employ compatible semantics and common interpretations of the information they exchange.

"Strong interoperability" implies that the organizations perform compatible, meaningful operations on exchanged information, and enforce compatible policies and procedures with regard to the exchanged information. Composability, a higher level concept, requires interoperability (in the common, narrow sense) as well as consistent, validated semantics. For results to be valid and meaningful, interoperable systems need to share common/compatible semantics. This requirement implies that interoperating systems must have compatible mechanisms for exchanging, representing, modifying, and updating semantic descriptions of the information. To the extent that the meaning and form of information is dynamic (i.e., can change over time), these systems should be able to dynamically alter their information processing mechanisms and, possibly, their representations.

"Designing In" versus "Retrofitting" Interoperability. Interoperability is achieved in one of two ways: It can be designed in at system inception, or it can be retrofitted into an existing system. Clearly, designing interoperability into a system is more effective, easier, and less expensive, given the cross-cutting nature of interoperability. Yet the need for interoperability is not always apparent when a system is designed, leading to the need to achieve interoperability after the fact. While retrofitting interoperability in this fashion may be worthwhile, it comes with enormous effort. The cross-cutting nature of interoperability requires it to be introduced into several aspects of a system (e.g., external interfaces, user interface, use of standards and communication protocols, internal processing, policies, and procedures for data handling; data access privileges). Investigating, and possibly modifying, these aspects of an existing system to make it interoperable will typically require substantial effort. Also, since interoperability depends on shared semantics, retrofitting a system to be interoperable can require major redesign of the system's representation and algorithms, as well as realignment of policies. These considerations can pose serious obstacles to retrofitting interoperability into existing systems.

Pros and Cons of Interoperability. Interoperability offers a number of advantages including: (a) *increased flexibility,* by allowing mixing and matching of systems; (b) *creation of new capabilities*, by composing new functions from existing ones; and (c) *increased cost-effectiveness*, by allowing reuse of existing systems and capabilities [Rothenberg, 2008]. The mixing and matching of systems enables performing unanticipated/unprecedented tasks from new combinations of existing functions. This can be accomplished on-the-fly (for nonrepetitive tasks), or *a priori* in a principled manner for repetitive or recurring tasks. Similarly, interoperability can reduce the cost of creating new capabilities by allowing existing systems to be reused in multiple ways for multiple purposes. An unheralded advantage of interoperability is that it hides overall system complexity from users by creating the illusion of an integrated system [Rothenberg, 2008]. These characteristics benefit users, developers, and customers alike. Interoperability is usually accomplished through a common user interface, uniform semantics, and uniform policies and procedures. However, in the real world this is not always the case. In some cases, interoperability is achieved through somewhat "clumsy" means. In fact, one might question if such systems can be claimed to be truly interoperable. An example of clumsy integration is the International Space Station, which employs a low-speed 1553 interface between itself and any experiment bolted on to it. While there is also a high-speed Ethernet interface for downlink from an experiment, there is no direct uplink path. Thus, if an experiment needs the high-speed link, crew members have to transfer the data on a thumb drive and walk it over to another computer that can perform data uplink over the Ethernet. While one could argue that this is an interoperable system, it is clearly not automated. The latter is implied by true interoperability.

Interoperability also has some disadvantages stemming from the increase in technical complexity and "open" system design [Rothenberg, 2008]. In particular, issues of privacy and security arise when systems are made interoperable. As important, costs can escalate from having to make systems interoperable. Finally, interoperability adds technical complexity to system design in that new interoperability requirements are now imposed that the designer has to satisfy. Even so, the benefits of interoperability invariably outweigh the costs.

Finally, the cross-cutting nature of interoperability has important implications. First, interoperability cannot be implemented piecemeal. Second, interoperability has a unique and crucial coordination role relative to the other cross-cutting concerns such as reliability, security, privacy, and information assurance [Rothenberg, 2008]. Third, interoperability cannot be added as an afterthought without incurring substantial costs and diminished effectiveness.

6. LEGACY SYSTEM INTEGRATION

Legacy systems are quite often critical organization assets that reflect the accumulated knowledge of an organization [Chowdhury and Iqbal, 2004]. Use of legacy systems primarily affects risk, structure, and integration resources in the SI ontology. Integrating, modernizing, or even replacing a legacy system can be fraught with risk. Whether in DoD, aerospace, or the commercial sector, legacy systems typically reflect a large number of changes that are made to them to continue to comply with changes in mission requirements or new business regulations or practices. Repeated modifications produce a cumulative effect on legacy system complexity. Primarily in defense but also in the commercial sector, a legacy system typically passes through multiple generations of developers and maintainers. This period could span decades for military systems that continue to be upgraded to satisfy new requirements. Legacy systems constitute massive DoD and corporate assets. Today, more than ever before, legacy systems are required to be integrated with other subsystems/applications within defense and commercial enterprises.

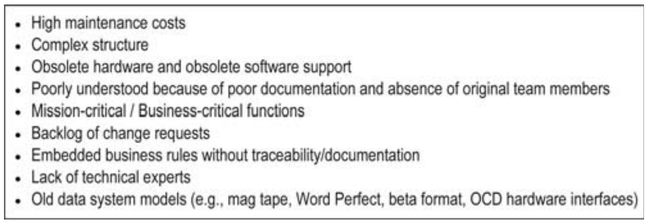
There are several challenges that need to be addressed when integrating or modernizing legacy systems. To begin with, there is rarely a complete specification of a legacy system. Legacy systems often tend to be inextricably intertwined with the business processes they support. Thus, if a legacy system is to be replaced, the impacted business processes also have to change, with potentially unpredictable costs and consequences. Also, important business rules may be embedded in the software and not documented anywhere. In general, legacy systems have several characteristics that increase their complexity (Table III).

The challenge when integrating legacy systems is to understand the functionality, design, operation, and performance of the system and to anticipate the types of changes required over the integration steps. It is important to realize that "legacy" does not have a negative connotation. In fact, an organization's knowledge capital often resides in legacy systems and legacy systems tend to work well. However, integrating legacy systems poses a challenge (Table III). The key issues to consider when integrating legacy systems are presented in Table IV. The issues raised in Table IV give rise to further concerns (e.g., format in which data are interchanged, how an application interprets messages sent to it; impact of changing a message definition on other applications/subsystems).

There are fundamentally two approaches to incorporating legacy systems: *reengineering*; and *integration*. *Reengineering* requires restructuring of a legacy system's code, which, in turn, requires that the system and code be well documented and/or can be automatically analyzed and transformed by an automatic process. *Integration* is faster and far less expensive than reengineering. Integration of a legacy system requires the definition of the roles of each subsystem, the interfaces for each subsystem, and the building of a "wrapper" for each subsystem (i.e., to encapsulate the system). An integration strategy can be intrusive ("white-box") or nonintrusive ("black-box"). A nonintrusive strategy requires knowledge of the external interfaces of the legacy system. A connection to an application is considered nonintrusive, if an existing entry/exit point is used. However, if the application's source code is modified, then the connection is considered intrusive. Intrusive connections are employed when custom code is developed to handle application needs or to increase performance. Nonintrusive connections are recommended if the information required from the application is already available from an existing interface and the transaction volume is low to moderate. Intrusive integration requires an initial reverse engineering process to gain an understanding of the internal system operation. After the code is analyzed and understood, intrusive integration often includes some restructuring of the system or code [Chowdhury and Iqbal, 2004].

Fundamentally, there are two major approaches for legacy systems integration: *application integration* and *data integration*. Application integration is based on the guiding philoso-

Table III. Characteristics of Legacy Systems (adapted from Chowdhury and Iqbal [2004])



- High maintenance costs
- Complex structure
- Obsolete hardware and obsolete software support
- Poorly understood because of poor documentation and absence of original team members
- Mission-critical / Business-critical functions
- Backlog of change requests
- Embedded business rules without traceability/documentation
- Lack of technical experts
- Old data system models (e.g., mag tape, Word Perfect, beta format, OCD hardware interfaces)

Table IV. Key Issues in Legacy System Integration

- Data Integration (several options)
  - identify and link records on the same subject (i.e., entity) in disparate systems
  - use metadata (different meanings in different applications) need custom interfaces between applications
  - perform data integration at the semantic level (i.e., based on actual content, not just the metadata)
- Connectivity
  - to each component in the architecture
- Routing
  - of messages between components
- Validation and Transformation
  - of data into and out of each application
- Interfacing with Each Application
  - based on each application's syntactic and semantic requirements
- Security
  - understanding of legacy system's security mechanisms and the vulnerabilities they pose
- Conformity
  - to organizational and business process structures
- Adaptations
  - the legacy system must be adapted to new business policies

phy that applications contain the business logic of the enterprise, and the solution lies in preserving that business logic by extending the application's interfaces to interoperate with other, occasionally newer, applications. Application integration solution categories include: user interface integration through technologies such as screen scraping (i.e., wrapping old text-based interfaces with new graphical interfaces), point-to-point integration (i.e., establishing communication channels between application pairs), message routing using various types of middleware (e.g., message-oriented middleware, CORBA, service-oriented architecture, and Web services). Data integration is motivated by the guiding philosophy that data are the real currency of the enterprise, and the implied business logic in the data and metadata can be easily manipulated directly by applications in the new architecture of the enterprise. Some popular data integration solutions are XML integration (for structured documents and data on the web) and data replication (i.e., the process of copying and maintaining database objects in multiple databases that make up a distributed database system). Replication provides users with fast, local access to shared data and greater availability to applications because alternative data access options exist. Even if one site becomes unavailable, users can continue to query, or even update, data at other locations. Database replication is often used to enable decentralized access to legacy data stored in mainframes.

7. HUMAN-SYSTEMS INTEGRATION

With the need to integrate humans with adaptable systems, human-system integration (HSI) has become a critical concern. Some of the HSI tools in current use include: concept mapping tools [Novak and Canas, 2008], cognitive function/task analysis [Roth et al., 2002], information and functional flow analysis (NASA SE Handbook [NASA, 2010], IMPRINT Pro [Alion, 2009], Jack [Badler et al., 1993], Job Assessment Software System (JASS) [Rossmeissl et al., 1983], Advisor 3.5 [Advisor 3.5 User Guide, 2000], Technique for Human Error Rate Prediction (THERP) [Swain, 1964]. These tools can help acquisition managers, program managers, and system engineers in articulating and documenting HSI issues for the acquisition community [Hale et al., 2009]. These tools are also a first step in understanding requirements (associated with operations), critical decisions, and workflow; identifying and appropriately allocating functions between humans and systems; understanding error and their likely consequences; and carrying out the trades required to optimize system effectiveness. However, these tools do not address more complex issues such as interactive Concept of Operations (CONOPS) development, changing attentional demands with multitasking and context switching; cognitive strategies in the face of sustained overload; and decision making in the face of ambiguity, uncertainty, risk, and stress [Madni, 2010, 2011]. New methods, processes, and tools are needed to enhance HSI capabilities when developing complex systems.

With the advent of new missions, CONOPS, tactics, and unprecedented, asymmetric threats, HSI has become a central issue in the military. The increased volume and pace of information in net-centric environments have created an information glut. Today, with increasing system autonomy, humans are being elevated from the role of primary job performers to that of supervisors. When it comes to unmanned autonomous platforms, humans are being increasingly called upon to manage and control multiple platforms, and accept multi-mission tasking. As a result, several new HSI issues have been identified. These include: information overload; dynamic attention allocation in multitasking and context switching situations; distributed decision making and team coordination; adaptable levels of automation; determining the right level of automation to ensure that human situation awareness and ability to back up automation in contingency situations are not impaired; and human decision biases in shared human-machine decision making. Designing systems with human considerations at the forefront of system development requires a disciplined systems approach and new methods, processes, and tools capable of addressing both human cognitive limitations and the self-adaptive nature of humans [Madni, 2010, 2011].

As systems continue to grow in scale and complexity, the role of the human is being redefined from that of an operator of a system to that of an agent within the system. Along with this change of human role comes the need for the human to continually adapt to changes in configuration, modes, and levels of system autonomy [Madni, 2010]. As such, human interactions drive SI structure, mechanisms, and system risks. The need for ongoing adaptation can often require multitasking and context switching that dramatically complicate the lives of humans [Madni, 2010]. This is not surprising in that while humans exhibit adaptivity, the adaptation tends to be slow, seldom complete, and occasionally not possible. Similarly, humans are poor at multitasking and context-switching, especially when context-switching frequency exceeds a maximum threshold [Madni, 2010, 2011]. While several advances have been made in human systems integration [Booher, 2003], existing methods, processes, and tools are woefully inadequate when it comes to integrating humans with adaptable systems [Madni, 2010, 2011]. Closed loop concept engineering is a promising approach that effectively addresses some of the limitations of existing methods and tools [Madni, 2012c]. Finally, several HSI measures have been identified in the literature including: percent time spent on collaboration and decision making; percent time spent on fixing errors; time to access/acquire necessary information; number of requests for information or clarification; number of times an operator is interrupted/disrupted when performing a task; and number of times humans express frustration, dissatisfaction, or confusion.

8. CONCLUDING REMARKS

Systems integration (SI) has become a key concern today as systems continue to grow in scale, complexity, and legacy content, and as humans continue to become increasingly more an integral part of the system [Madni, 2010; Brown and Eremenko, 2009; Rodriguez and Merseguer, 2010]. In this paper, we discussed the key challenges facing SI today, along with examples of SI failures. We presented an SI ontology that standardizes SI concepts and informs the creation of integration models that reflect integration requirements and constraints. The SI ontology serves a number of purposes. These include: establishing a common terminology among stakeholders; defining both technical and nontechnical aspects of SI; and supporting reasoning to detect interface inconsistencies and errors that are the most common causes of SI failure. After that, we critically examined factors such as interoperability, legacy systems, and human-system interactions and their impact on SI. We then discussed how these considerations can be introduced in the SI model (i.e., ontology). It is our hope that this paper will serve both as a tutorial and a starting point for systems engineers who have the responsibility for planning and executing SI initiatives.

REFERENCES

D.J. Abel, P.J. Kilby, and J.R. Davis, "The Systems Integration Problem," International Journal of Geographical Information Systems, 8:1, 1-12. 1994.

Advisor 3.5 User's Guide, BNH Expert Software, 2000.

Alion Science and Technology, Boulder, CO, IMPRINT Pro user guide, Version 3.0, Volume 2, March 2009.

N.I. Badler, C.B. Phillips, and B.L. and Webber, Simulating humans, Computer graphics, animation and control, Oxford University Press, Oxford, 1993.

M. Best, The new competitive advantage: The renewal of American industry, Oxford, University Catalogue Press, 2001.

S. Biffl, A. Schatten, and A. Zoitl, Integration of heterogeneous engineering environments for the automation systems lifecycle, 7th IEEE Int Conf Indust Inform, INDIN 2009, 2009, pp. 576–581.

M. Bonjour and G. Falquet, "Concept bases: A support to information systems integration," in Wijers, G., Brinkkemper, S., Wasserman, T. (Eds.), Advanced Information Systems Engineering, Lecture Notes in Computer Science 811, Springer, New York, 1994, pp. 242–255.

H.R. Booher (Editor), Handbook of human systems integration, Wiley Online Laboratory, Hoboken, NJ, 2003.

O.C. Brown, P. Eremenko, and P.D. Conopy, Value-centric design methodologies for fractionated spacecraft: AIAA SPACE Conference, 2009.

T.R. Browning, Applying the design structure matrix to system decomposition and integration problems: A review and new directions, IEEE Trans Eng Management 48(3) (August 2001).

G. Cabri, Agent-based plug-and-play integration of role-enabled medical devices, Joint Workshop High Confidence Med Devices Software Syst Med Device Plug-and-Play Interoperability, HCMDSS-MDPnP, June 2007, pp. 25–27.

D. Chen, G. Doumeingts, and F. Vernadat, Architectures for enterprise integration and interoperability: Past, present and future, Comput Indust 59(7) (2008), 647–659.

M.W. Chowdhury and M.Z. Iqbal, Integration of legacy systems in software architecture, SAVCBS 2004 Specification and Verification of Component-Based Systems 110.

F.A. Cummins, Enterprise integration: An architecture for enterprise application and systems engineering, Wiley, Hoboken, NJ, 2002.

P.K. Davis and R.H. Anderson, Improving the composability of Department of Defense models and simulations, The Journal of Defense Modeling and Simulation: Applications, Methodology, and Technology, 1.1 (2004), 5–17.

C. Hale, C. Ching, B. Brett, and A. Rothblum, Survey of Human-Systems Integration (HSI) tools for USCG acquisition, Report No. CG-D-04-09, US Coast Guard, US Department of Homeland Security, Washington, DC, April 2009.

A.Y. Halevy, N. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka, Enterprise information integration: Successes, challenges and controversies, Proc 2005 ACM SIGMOD Int Conf Management Data, SIGMOD '05, 2005, pp. 778–787.

M. Hobday, A. Davies, and A. Prencipe, Systems integration: A core capability of the modern corporation, Indust Corp Change 14(6) (2005), 1109–1143.

B. Horowitz, J. Liebman, C. Ma, T.J. Koo, A. Sangiovanni-Vincentelle, and S.S. Sastry, Platform-based embedded software design and system integration for autonomous vehicles, Proc IEEE 91(1) (2003), 198–211.

M.O. Khan, M. Sievers, and S. Standley, Model-based verification and validation of spacecraft avionics, Proceedings of the AIAA Infotech@Aerospace Conf, 2012.

C. Kilroy, Investigations: Air France 296, AirDisaster.com, June 22, 2006.

V. Kotov, Systems of systems as communicating structures, Hewlett Packard Laboratories, 1997.

A.J. Krygiel, Behind the wizard's curtain: An integration environment for a system of systems, CCRP, July 1999.

J. Lee, K. Siau, and S. Hong, Enterprise integration with ERP and EAI, Commun ACM 46(2) (February 2003), 54–60.

J. Luo, Z. Shi, M. Wang, and J. Hu, "AGrIP: An agent grid intelligent platform for distributed system integration," in Advanced Web and Network Technologies and Applications, Lecture Notes in Computer Science 3842, 2006, pp. 590–594.

A.M. Madni, Integrating humans with software and systems: Technical challenges and a research agenda, Syst Eng 13(3) (2010), 232–245.

A.M. Madni, Integrating humans with and within software and systems: Challenges and opportunities (Invited Paper), CrossTalk J Defense Software Eng (May/June 2011), "People Solutions."

A. Madni, Adaptable platform-based engineering: Key enablers and outlook for the future, Syst Eng 15(1) (2012a), 97–107.

A.M. Madni, Elegant systems design: Creative fusion of simplicity and power, Syst Eng 15(3) (2012a).

A.M. Madni, Affordable, adaptive and effective: The case for engineering resilient systems, Invited Provocative Talk, Workshop Resilient Syst, Keck Institute for Space Studies, 2012b.

A.M. Madni and S. Jackson, Towards a conceptual framework for resilience engineering, Systems Journal, IEEE, 3(2), 181–191.

M.W. Maier and E. Rechtin, The art of systems architecting, 3rd edition, CRC Press, 2002, Boca Raton, FL, 2009.

I. Mayk and A.M. Madni, The role of ontology in system-of-systems acquisition, Proc 2006 Command Control Res Technol Symp, San Diego, CA, June 20–22, 2006.

P.J. Mitchell, M.L. Cummings, and T.B. Sheridan, Human supervisory control issues in network centric warfare, Massachusetts Institute of Technology, Human and Automation Laboratory, Cambridge, MA, 2004.

P.J. Mosterman, J. Ghidella, and J. Friedman, Model-based design for system integration, Proceedings of the Canadian Engineering Education Association, 2011, Conf Des, 2005.

NASA System Engineering Handbook, DIANE Publishing, 2010.

Y. Naudet, T. Latour, W. Guedria, and D. Chen, Toward a systemic formalization of interoperability, Comput Indust 61 (2010), 176–185.

R. Neches, The challenges of engineering in the 21st century, Collaboration Technologies and Systems (CTS), 2012 International Conference, IEEE, pp. 367–370.

R. Neches and A.M. Madni, Towards affordably adaptable and effective systems, Syst Eng 15(1) (2012).

E.G. Nilsson, E.K. Nordhagen, and G. Oftedal, Aspects of systems integration, Proc First Int Conf Syst Integration, 1990, pp. 434–443.

J.D. Novak and A.J. Canas, The theory underlying concept MAPS and how to construct and use them, Institute for Human and Machine Cognition, November 2008.

NRC, Defense modeling, simulation and analysis: Meeting the challenge, Committee on Modeling and Simulation for Defense Transformation, National Research Council, Washington, DC, 2006.

A. Prencipe, A. Davies, and M. Hobday (Editors), The business of systems integration, Oxford University Press, Oxford, 2011.

D. Rehage, U. Caol, A. Merkel, and A. Vahl, "The effects on reliability of aircraft systems based on integrated modular avionics," in Computer Safety, Reliability, and Security, Lecture Notes in Computer Science 3219, Springer, New York, 2004, pp. 224–238.

R.J. Rodriguez and J. Merseguer, "Integrating fault-tolerant techniques into the design of critical systems," in Architecting critical Systems, Lecture Notes in Computer Science 6150, Springer, New York, 2010, pp. 33–51.

P.G. Rossmeissl, B.W. Tillman, K.E. Rigg, and P.R. Best, Job Assessment Software System (JASS) for analysis of weapon systems personnel requirements, No. MGA-4581-MRO McFann Gray and Associates, Inc., Carmel, CA, 1983. Research Report 1355, Army Research Institute, November 1983.

W. Rossak and P.A. Ng, Some thoughts on systems integration: A conceptual framework, J Syst Integration 1 (1991), 97–114.

E.M. Roth, E.S. Patterson, and R.J. Mumaw, "Cognitive engineering: issues in user-centered system design," in J.J. Marciniak (Editor), Encyclopedia of Software Engineering, 2nd edition, Wiley-Interscience, Hoboken, NJ, 2002, pp. 163–179.

J. Rothenberg, Interoperability as a cross-cutting concern, Interoperabilitect: Eerlijk Zullen we ulles delen, 2008.

A.P. Sage and C.L. Lynch, Systems integration and architecting: An overview of principles, practices, and perspectives, Syst Eng 1 (1998), 176–227.

C.E. Siemieniuch and M.A. Sinclair, Systems integration, Appl Ergonom 37 (2006), 91–110.

V. Stavridou, Integration in software intensive systems, J Syst Software 48 (1999), 91–104.

A.D. Swain, THERP, SC-R-64-1338, Sandia National Laboratories, Albuquerque, NM, August 1964.

M. Vuletid, L. Pozzi, and P. Ienne, Seamless hardware-software integration in reconfigurable computing systems, Des Test Comput, IEEE 22 (2005), 102–113.

Azad Madni is a Professor in the Daniel J. Epstein Department of Industrial and Systems Engineering and the Director of the multidisciplinary Systems Architecting and Engineering (SAE) Program in the University of Southern California's Viterbi School of Engineering in Los Angeles, CA. He holds joint appointments in USC's Keck School of Medicine and Rossier School of Education. He is the founder, Chairman, and Chief Scientist of Intelligent Systems Technology, Inc. He received his B.S., M.S., and Ph.D. degrees from the University of California, Los Angeles. His research has been sponsored by several major government research agencies including OSD, DARPA, DHS S&T, MDA, DTRA, ONR, AFOSR, AFRL, ARI, RDECOM, NIST, DOE, and NASA. He is the recipient of the 2011 Pioneer Award from the International Council of Systems Engineering. In 2012, he received INCOSE-LA's Exceptional Achievement Award for transformative advances in systems engineering. He is also the recipient of SBA's 1999 National Tibbetts Award for California, the 2000 Blue Chip Entrepreneurship Award from MassMutual and the U.S. Chamber of Commerce, the 2008 President's Award and the 2006 C.V. Ramamoorthy Distinguished Scholar Award from the Society of Design and Process Science. He is also the recipient of the Developer of the Year Award from the Technology Council of Southern California in 2000 and 2004. He is a Life Fellow of IEEE and IETE, and a Fellow of AAAS, INCOSE, AIAA, and SDPS. He is listed in the major *Who's Who* including *Marquis Who's Who in Science and Engineering*, *Who's Who in Industry and Finance*, and *Who's Who in America*.

Michael Sievers is a Senior Systems Engineer at Caltech's Jet Propulsion Laboratory, Pasadena, CA and a Lecturer in System Engineering at the University of Southern California, Los Angeles, CA. He conducts research in model-based systems engineering as well as contributing to a number of the Jet Propulsion Lab's flight missions. He holds a Ph.D. in Computer Science from the University of California, Los Angeles and is a member of INCOSE and a Senior Member of the IEEE and AIAA.